**Введение**

Это техническое руководство поможет начинающим разработчикам создать Telegram-бота «Поддержка рядом», предназначенного для помощи людям с ограниченными возможностями здоровья (ОВЗ) в управлении приёмом лекарств. Бот позволяет добавлять лекарства, устанавливать расписание, получать напоминания, подтверждать или откладывать приём, генерировать отчёты и уведомлять доверенных лиц о пропуске приёма. Руководство включает пошаговые инструкции и примеры кода из файла bot.py. Бот разработан в рамках проекта «Помощь людям с ОВЗ: сайт для оказания экстренной помощи» и ориентирован на доступность и простоту.

**Требования**

1. Установленные программы:

1. Python 3.8 или выше (скачать с https://www.python.org/).

2. Текстовый редактор (например, Visual Studio Code).

2. Библиотеки Python:

1. python-telegram-bot (версия 13.x) — для работы с Telegram Bot API.

2. pandas — для экспорта данных в Excel.

3. openpyxl — для работы с файлами Excel.

3. Аккаунт Telegram:

1. Создать бота через @BotFather для получения токена.

**Архитектура бота**

1. Входной интерфейс: Обработка команд и сообщений через Telegram Bot API с использованием python-telegram-bot.

2. Обработчик диалогов: Управление последовательным вводом данных (название, дозировка и т.д.) через ConversationHandler.

3. Планировщик задач: Отправка напоминаний в заданное время с помощью JobQueue.

4. Хранилище данных: Временное хранение данных о лекарствах и пользователях в словарях Python.

5. Генерация отчётов: Создание отчётов за неделю/месяц и экспорт в Excel с использованием pandas и openpyxl.

6. Уведомления: Отправка сообщений доверенным лицам при пропуске приёма через Telegram API.

7. Логирование: Запись ошибок и событий для отладки с помощью модуля logging.

**Пошаговые инструкции**

1. Установка окружения

1. Убедитесь, что Python установлен:

1. Откройте терминал и выполните:

```

python --version

```

2. Если Python не установлен, скачайте и установите с https://www.python.org/.

2. Установите необходимые библиотеки:

1. Выполните в терминале:

```

pip install python-telegram-bot pandas openpyxl

```

3. Установите текстовый редактор:

1. Рекомендуется Visual Studio Code (скачать с https://code.visualstudio.com/).

2. Создание бота в Telegram

1. Откройте Telegram и найдите @BotFather:

1. Отправьте команду /start.

2. Отправьте команду /newbot.

2. Следуйте инструкциям @BotFather:

1. Задайте имя бота (например, «SupportBot»).

2. Задайте username (например, @SupportBot).

3. Сохраните токен:

1. @BotFather выдаст токен, например, 8156244486:AAF2VLO95xxJQbp6hsm7dBw-YdD2B-XoYO4.

2. Скопируйте токен в безопасное место.

3. Настройка проекта

1. Создайте папку проекта:

1. Например, support\_bot.

2. Создайте файл bot.py:

1. В папке проекта создайте файл bot.py с помощью текстового редактора.

3. Скопируйте базовый код:

1. Используйте код ниже для создания минимальной версии бота.

4. Реализация базового функционала

1. Инициализация бота:

1. Настройте Updater и логирование:

```python

import logging

from telegram.ext import Updater

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

```

2. Обработка команды /start:

1. Отобразите главное меню с кнопками:

```python

def start(update, context):

keyboard = [[InlineKeyboardButton("Добавить лекарство", callback\_data='add\_med')]]

reply\_markup = InlineKeyboardMarkup(keyboard)

update.message.reply\_text("Привет! Я бот для напоминания о лекарствах. Выбери действие:", reply\_markup=reply\_markup)

```

3. Диалог добавления лекарства:

1. Используйте ConversationHandler для ввода названия и времени:

```python

ConversationHandler(entry\_points=[CallbackQueryHandler(add\_med\_start, pattern='^add\_med$')], ...)

```

4. Напоминания:

1. Запланируйте ежедневные уведомления:

```python

context.job\_queue.run\_daily(send\_reminder, datetime.time(hour=hour, minute=minute), ...)

```

5. Отображение списка лекарств:

1. Покажите добавленные лекарства:

```python

def show\_my\_meds(update, context):

text = "Ваши лекарства:\n"

```

Базовый код (bot.py)

```python

import logging

from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup

from telegram.ext import Updater, CommandHandler, CallbackQueryHandler, MessageHandler, Filters, ConversationHandler, CallbackContext

from datetime import datetime

Настройка логирования

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

Состояния для добавления лекарства

ADD\_MED\_NAME, ADD\_MED\_TIME = range(2)

Хранилище данных

medications = {}

def start(update: Update, context: CallbackContext) -> None:

"""Команда /start — показывает главное меню."""

keyboard = [

[InlineKeyboardButton("Добавить лекарство", callback\_data='add\_med')],

[InlineKeyboardButton("Мои лекарства", callback\_data='my\_meds')],

]

reply\_markup = InlineKeyboardMarkup(keyboard)

update.message.reply\_text("Привет! Я бот для напоминания о лекарствах. Выбери действие:", reply\_markup=reply\_markup)

def add\_med\_start(update: Update, context: CallbackContext) -> int:

"""Начало добавления лекарства."""

update.callback\_query.answer()

update.callback\_query.edit\_message\_text("Введите название лекарства:")

return ADD\_MED\_NAME

def add\_med\_name(update: Update, context: CallbackContext) -> int:

"""Сохранение названия лекарства."""

context.user\_data['med\_name'] = update.message.text

update.message.reply\_text("Введите время приёма (например, 9:00):")

return ADD\_MED\_TIME

def add\_med\_time(update: Update, context: CallbackContext) -> int:

"""Сохранение времени приёма и завершение."""

user\_id = update.effective\_user.id

med\_name = context.user\_data['med\_name']

med\_time = update.message.text

if user\_id not in medications:

medications[user\_id] = []

medications[user\_id].append({'name': med\_name, 'time': med\_time})

update.message.reply\_text("Лекарство добавлено!", reply\_markup=InlineKeyboardMarkup([

[InlineKeyboardButton("В главное меню", callback\_data='back')]

]))

Запланировать напоминание

hour, minute = map(int, med\_time.split(':'))

context.job\_queue.run\_daily(

send\_reminder,

datetime.time(hour=hour, minute=minute),

context={'user\_id': user\_id, 'med\_name': med\_name}

)

context.user\_data.clear()

return ConversationHandler.END

def send\_reminder(context: CallbackContext):

"""Отправка напоминания."""

job = context.job

user\_id = job.context['user\_id']

med\_name = job.context['med\_name']

context.bot.send\_message(

chat\_id=user\_id,

text=f"⏰ Напоминание: примите {med\_name}!"

)

def show\_my\_meds(update: Update, context: CallbackContext):

"""Показать список лекарств."""

user\_id = update.effective\_user.id

user\_meds = medications.get(user\_id, [])

if not user\_meds:

update.callback\_query.edit\_message\_text(

text="Нет добавленных лекарств.",

reply\_markup=InlineKeyboardMarkup([[InlineKeyboardButton("В главное меню", callback\_data='back')]])

)

return

text = "Ваши лекарства:\n"

for med in user\_meds:

text += f"- {med['name']} в {med['time']}\n"

update.callback\_query.edit\_message\_text(

text=text,

reply\_markup=InlineKeyboardMarkup([[InlineKeyboardButton("В главное меню", callback\_data='back')]])

)

def button\_handler(update: Update, context: CallbackContext) -> None:

"""Обработка кнопок."""

query = update.callback\_query

query.answer()

if query.data == 'add\_med':

return add\_med\_start(update, context)

elif query.data == 'my\_meds':

show\_my\_meds(update, context)

elif query.data == 'back':

query.edit\_message\_text(

text="Выбери действие:",

reply\_markup=InlineKeyboardMarkup([

[InlineKeyboardButton("Добавить лекарство", callback\_data='add\_med')],

[InlineKeyboardButton("Мои лекарства", callback\_data='my\_meds')],

])

)

def main():

"""Запуск бота."""

updater = Updater("YOUR\_BOT\_TOKEN\_HERE") Замените на ваш токен

dispatcher = updater.dispatcher

dispatcher.add\_handler(CommandHandler("start", start))

add\_med\_conv = ConversationHandler(

entry\_points=[CallbackQueryHandler(add\_med\_start, pattern='^add\_med$')],

states={

ADD\_MED\_NAME: [MessageHandler(Filters.text & ~Filters.command, add\_med\_name)],

ADD\_MED\_TIME: [MessageHandler(Filters.text & ~Filters.command, add\_med\_time)],

},

fallbacks=[]

)

dispatcher.add\_handler(add\_med\_conv)

dispatcher.add\_handler(CallbackQueryHandler(button\_handler))

updater.start\_polling()

updater.idle()

if \_\_name\_\_ == '\_\_main\_\_':

main()

```

1. Замените токен:

1. В строке updater = Updater("YOUR\_BOT\_TOKEN\_HERE") вставьте токен от @BotFather.

2. Сохраните файл:

1. Сохраните код в bot.py в папке проекта.

3. Запустите бота:

1. В терминале перейдите в папку проекта:

```

cd support\_bot

```

2. Выполните:

```

python bot.py

```

5. Тестирование базового функционала

1. Запуск бота:

1. Отправьте /start в Telegram, чтобы получить меню с кнопками «Добавить лекарство» и «Мои лекарства».

2. Добавление лекарства:

1. Нажмите «Добавить», введите «Парацетамол» и «9:00», чтобы получить сообщение «Лекарство добавлено!».

3. Просмотр списка:

1. Нажмите «Мои лекарства», чтобы увидеть «Парацетамол в 9:00».

4. Напоминание:

1. Установите время на текущую минуту +1, подождите, чтобы получить сообщение «⏰ Напоминание: примите...».

1. Проверьте команду /start:

1. Откройте Telegram, найдите бота, отправьте /start.

2. Добавьте лекарство:

1. Нажмите «Добавить лекарство», введите название и время.

3. Проверьте список:

1. Нажмите «Мои лекарства», убедитесь, что лекарство отображается.

4. Проверьте напоминание:

1. Для теста установите время на ближайшую минуту, подождите уведомления.

6. Расширение функционала

1. Дозировка и форма:

1. Добавьте ввод дозировки и выбор формы лекарства:

```python

def add\_med\_dosage(update: Update, context: CallbackContext) -> int:

context.user\_data['med\_dosage'] = update.message.text

reply\_keyboard = [['Таблетки', 'Капсулы'], ['Инъекции', 'Сироп'], ['Другое']]

update.message.reply\_text(

"Выберите форму лекарства:",

reply\_markup=ReplyKeyboardMarkup(reply\_keyboard, one\_time\_keyboard=True)

)

return ADD\_MED\_FORM

```

2. Подтверждение приёма:

1. Добавьте кнопку для подтверждения и сохранения истории:

```python

InlineKeyboardButton("Я принял(а) лекарство", callback\_data=f'taken\_{med\_name}')

```

3. Отсрочка напоминания:

1. Реализуйте отсрочку на 10 минут:

```python

context.job\_queue.run\_once(send\_reminder, when=10\*60, ...)

```

4. Отчёты:

1. Сгенерируйте отчёты за 7/30 дней:

```python

def generate\_report(user\_id: int, days: int) -> str:

user\_meds = medications.get(user\_id, [])

if not user\_meds:

return "Нет данных для отчета."

report = f"Отчет за последние {days} дней:\n\n"

for med in user\_meds:

recent\_history = [h for h in med['history'] if h['date'] > datetime.now() - timedelta(days=days)]

taken = sum(1 for h in recent\_history if h['status'] == 'принято')

missed = sum(1 for h in recent\_history if h['status'] == 'пропущен')

total = taken + missed

compliance = (taken / total) \* 100 if total > 0 else 0

report += f"💊 {med['name']}:\n✅ Принято: {taken} раз\n❌ Пропущено: {missed} раз\n📊 Соблюдение: {compliance:.1f}%\n\n"

return report

```

5. Экспорт в Excel:

1. Экспортируйте историю приёма в .xlsx:

```python

import pandas as pd

def export\_to\_excel(user\_id: int) -> str:

user\_meds = medications.get(user\_id, [])

if not user\_meds:

return None

data = [{'Лекарство': med['name'], 'Дата': event['date'], 'Статус': event['status']}

for med in user\_meds for event in med['history']]

df = pd.DataFrame(data)

filename = f"report\_{user\_id}.xlsx"

df.to\_excel(filename, index=False)

return filename

```

6. Доверенные лица:

1. Уведомляйте при пропуске через 30 минут:

```python

def notify\_relatives(context: CallbackContext, user\_id: int, med\_name: str):

for relative in relatives.get(user\_id, []):

try:

context.bot.send\_message(

chat\_id=relative,

text=f"⚠️ Пользователь не подтвердил прием {med\_name} в назначенное время!"

)

except Exception as e:

logger.error(f"Не удалось уведомить {relative}: {e}")

```

1. Добавьте дозировку и форму:

1. Расширьте ConversationHandler, добавив состояния ADD\_MED\_DOSAGE, ADD\_MED\_FORM.

2. Реализуйте подтверждение и отсрочку:

1. В функции send\_reminder добавьте кнопки.

3. Добавьте отчёты:

1. Реализуйте функцию generate\_report.

4. Реализуйте экспорт:

1. Используйте pandas для создания Excel-файла.

5. Добавьте уведомления доверенных лиц:

1. Реализуйте функцию notify\_relatives.

7. Обеспечение доступности

1. Простой интерфейс:

1. Минимизируйте количество шагов, используйте чёткие текстовые подсказки.

2. Экранные читалки:

1. Применяйте текстовые сообщения с эмодзи (⏰, ✅), совместимые с NVDA.

3. Предустановленные варианты:

1. Настройте частоту: «1 раз в день» → 9:00, «2 раза» → 9:00, 21:00.

1. Упростите ввод:

1. Используйте ReplyKeyboardMarkup для выбора вариантов:

```python

reply\_keyboard = [['1 раз в день', '2 раза в день'], ['3 раза в день', 'По часам']]

```

2. Поддержите экранные читалки:

1. Добавляйте эмодзи и чёткие тексты в сообщения.

8. Тестирование расширенного функционала

1. Полное добавление:

1. Введите название, дозировку, форму, частоту, время, чтобы лекарство сохранилось и напоминание запланировалось.

2. Подтверждение приёма:

1. Нажмите «Я принял(а)» в напоминании, чтобы получить сообщение «✅ Прием подтвержден!».

3. Отсрочка:

1. Нажмите «Отложить», подождите 10 минут, чтобы получить повторное напоминание.

4. Отчёт за неделю:

1. Выберите «Отчет за неделю», чтобы увидеть отчёт с процентом соблюдения.

5. Экспорт:

1. Выберите «Экспорт данных», чтобы получить файл .xlsx в чате.

6. Уведомление родственников:

1. Пропустите подтверждение, подождите 30 минут, чтобы сообщение отправилось доверенному лицу.

1. Проверьте полный процесс добавления:

1. Добавьте лекарство с дозировкой, формой и частотой.

2. Проверьте напоминания:

1. Подтвердите или отложите напоминание.

3. Проверьте отчёты и экспорт:

1. Сгенерируйте отчёт, экспортируйте данные.

4. Проверьте уведомления:

1. Добавьте доверенное лицо, пропустите напоминание.

9. Рекомендации для начинающих

1. Изучите документацию:

1. Telegram Bot API: https://core.telegram.org/bots/api

2. python-telegram-bot: https://python-telegram-bot.readthedocs.io/

2. Тестируйте постепенно:

1. Добавляйте функции по одной, проверяйте их работу.

3. Используйте логирование:

1. Добавьте logging.info("Сообщение") для отладки.

4. Планируйте масштабирование:

1. Замените словари на SQLite для реального проекта:

```python

import sqlite3

conn = sqlite3.connect('medications.db')

```

Процесс обработки запросов

1. Получение сообщения:

1. Пользователь отправляет /start или нажимает кнопку, используя Updater и Telegram API.

2. Обработка команды:

1. Вызывается функция start или button\_handler через CommandHandler и CallbackQueryHandler.

3. Диалог:

1. Последовательный ввод данных через состояния в ConversationHandler.

4. Планирование:

1. Создание задачи напоминания с помощью JobQueue.

5. Отправка уведомления:

1. Отправка сообщения в заданное время через Telegram API.

6. Сохранение данных:

1. Запись данных в словарь или базу (в будущем SQLite).

7. Генерация отчёта:

1. Расчёт статистики и экспорт с использованием pandas и openpyxl.

**Заключение**

Это руководство охватывает создание Telegram-бота «Поддержка рядом» от базового функционала до расширенных функций, таких как отчёты и уведомления доверенных лиц. Бот разработан с учётом доступности для людей с ОВЗ, используя простой интерфейс и текстовые сообщения, совместимые с экранными читалками. Для дальнейшего развития рекомендуется:

1. Перейти на SQLite для хранения данных.

2. Добавить голосовой ввод для пользователей с нарушениями моторики.

3. Интегрировать с внешними системами (например, медицинскими API).

Исходный код доступен в bot.py, документация — в bot\_documentation.md.